

NY Car Crashes and Weather Conditions

User Information & Guidelines

## 

## Data Engineering 2

#### Students

Yuri Almeida – 1901587

John Joyce – 2001928

Istvan Janco – 2003877

Table of Contents

[Project and Data Overview 3](#_Toc58533522)

[Overview 3](#_Toc58533523)

[Data Sources 3](#_Toc58533524)

[Analytics Questions 3](#_Toc58533525)

[Motor Vehicle Accidents 4](#_Toc58533526)

[Precipitation Data 12](#_Toc58533527)

[Analytics 20](#_Toc58533528)

[Join + Fix Data Types 21](#_Toc58533529)

[Analysis: Histograms 23](#_Toc58533530)

[Analysis: Maps 24](#_Toc58533531)

[Permanent Data Storage 27](#_Toc58533532)

[MySQL Server 28](#_Toc58533533)

[Conclusion 30](#_Toc58533534)

[Project / Tool Overview 30](#_Toc58533535)

## Project and Data Overview

### Overview

The following project was developed for the Data Engineering 2 course of the Central European University. The main idea is to provide monthly information regarding the motor vehicle accidents and the weather conditions at New York city. Using the software KNIME and through API`s, it was a created a workflow to retrieve, store and quickly analyze the datasets.

### Data Sources

The data sources are a combination between:

* Weather - National Centers for Environmental Information (NCDC), National Oceanic and Atmospheric Administration (NOAA): <https://www.ncdc.noaa.gov/cdo-web/webservices/v2#datasets>
* Motor Vehicle Accidents - NYC Open Data Source: https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95

### Analytics Questions

#### Motor Vehicle Stats

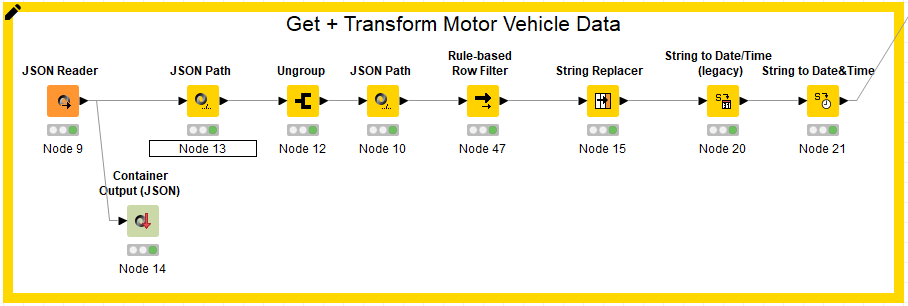
* When did those accidents happen? Where? Which Street?
* What were the vehicles involved? What were the contribution factors for that?
* How many were injured? Died? Cyclists? Pedestrians? Drivers?

#### Weather Stats

* What was the precipitation rate on a given day?

### Motor Vehicle Accidents

#### KNIME Workflow



#### Data Description

* **crash\_date:** Occurrence date of collision
* **crash\_time:** Occurrence time of collision
* **on\_street\_name:** Street on which the collision occurred
* **contributing\_factor\_vehicle\_1:** Factors contributing to the collision for designated vehicle
* **contributing\_factor\_vehicle\_2:** Factors contributing to the collision for designated vehicle
* **collision\_id:** Unique record code generated by system. Primary Key for Crash table.
* **vehicle\_type\_code1:** Type of vehicle based on the selected vehicle category (ATV, bicycle, car/suv, ebike, escooter, truck/bus, motorcycle, other)
* **vehicle\_type\_code2:** Type of vehicle based on the selected vehicle category (ATV, bicycle, car/suv, ebike, escooter, truck/bus, motorcycle, other)

**Number\_of**

\_People:

\_Cyclist:

\_Pedestrians:

\_Motorists:

* **\_killed:** Number of killed
* **\_injured:** Number of injured

#### Workflow Description

1. **JSON Reader (Node 9):** Retrieves the data into JSON format.

**Query Parameters** (Ex: March)

[https://data.cityofnewyork.us/resource/h9gi-nx95.json?$where=crash\_date between '2019-05-01T00:00:00' and '2019-05-31T00:00:00'&$limit=20000&$$app\_token=Huc1YV43ApZmVtseIyY5II8ka](https://data.cityofnewyork.us/resource/h9gi-nx95.json?$where=crash_date%20between%20'2019-05-01T00:00:00'%20and%20'2019-05-31T00:00:00'&$limit=20000&$$app_token=Huc1YV43ApZmVtseIyY5II8ka)

**Data Range**

$where=**crash\_date** between **'2019-05-01T00:00:00**' and **'2019-05-31T00:00:00'**

**Search Limit**

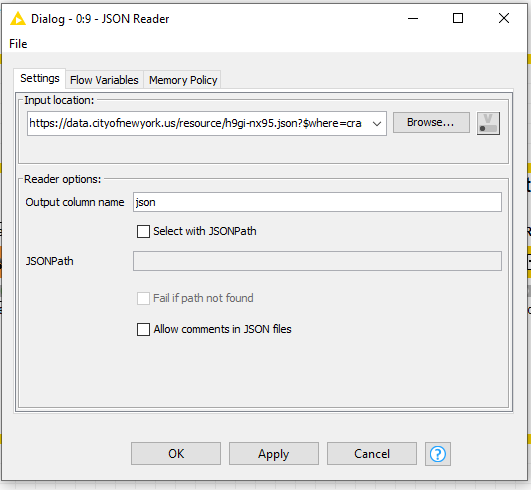
$**limit**=**20000**

**App Token**

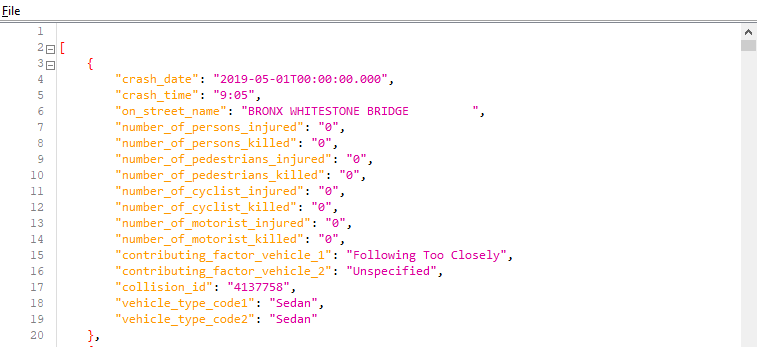
$app\_token=Huc1YV43ApZmVtseIyY5II8ka

**PS: It’s important to mention that the token creation it´s really necessary. Otherwise, the program will face a limitation of number of observations while running the query**

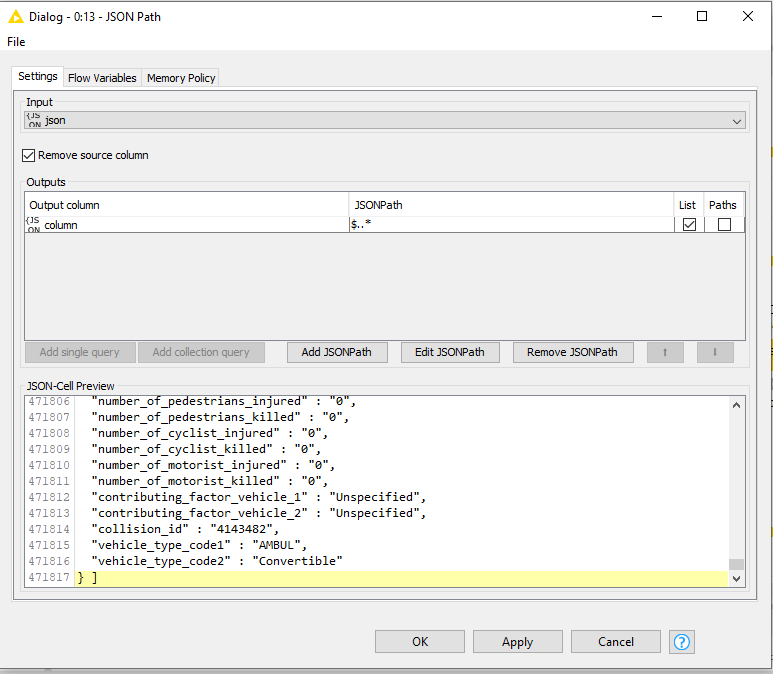
For more information, regarding the API, token generation and how to determine the parameters of the query please access: [*https://dev.socrata.com/foundry/data.cityofnewyork.us/h9gi-nx95*](https://dev.socrata.com/foundry/data.cityofnewyork.us/h9gi-nx95)



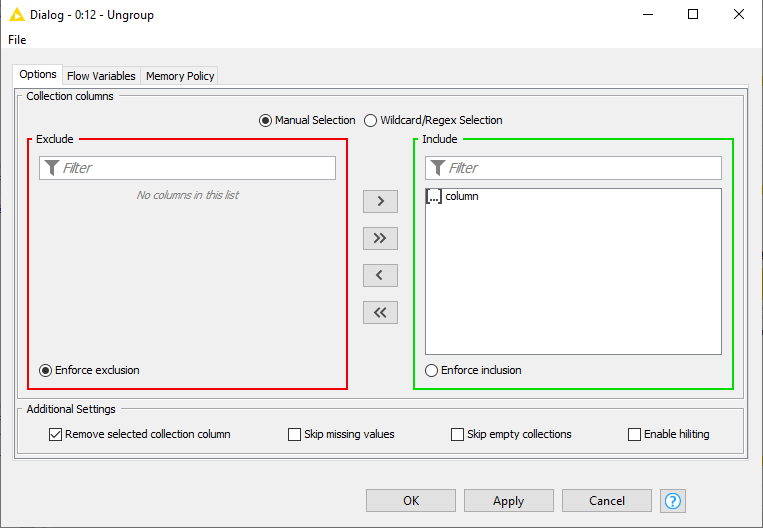
1. **Container Output (JSON) (Node 14):** Check the JSON output.

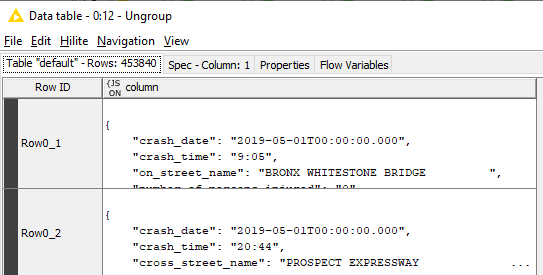


1. **JSON Path (Node 13):** read JSON format and generate the column/observation to be ungrouped

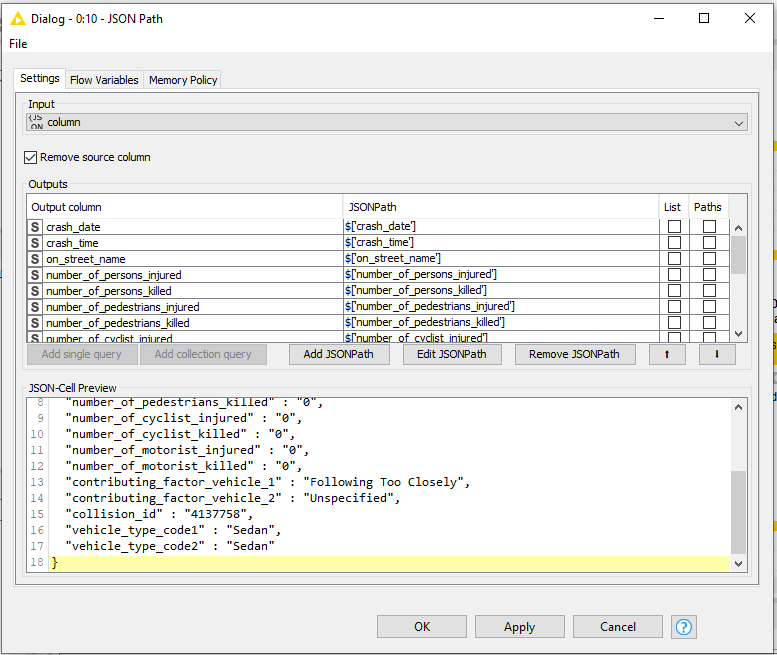


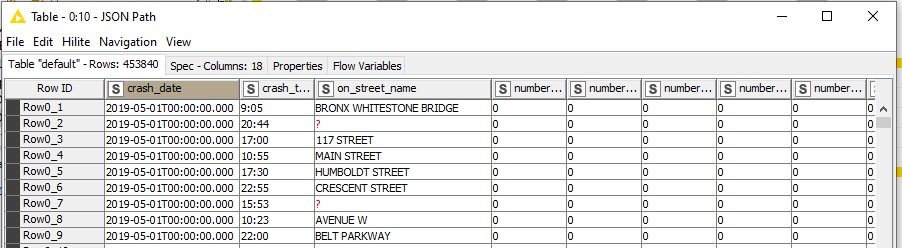
1. **Ungroup (Node 12):** ungroup the JSON unique row/column into different observations (rows)



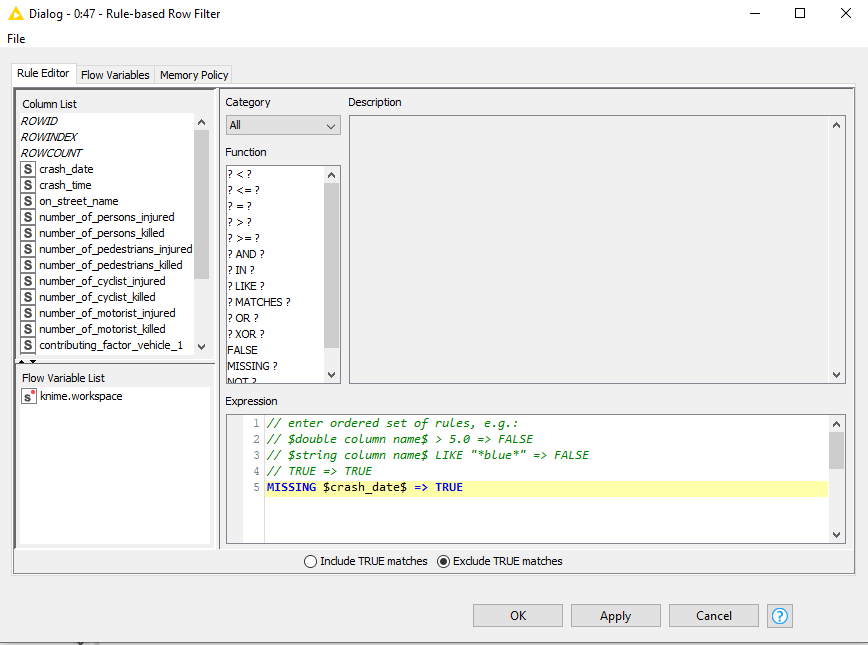


1. **JSON Path (Node 10):** separate now the unique column to different ones

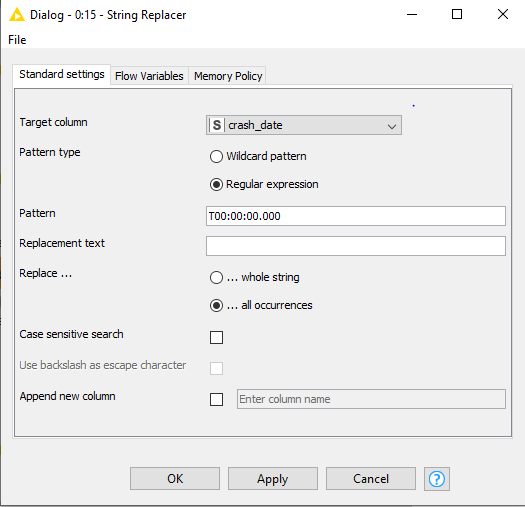




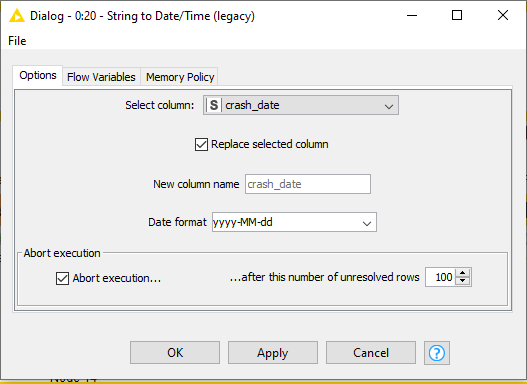
1. **Rule-Based Row Filter (Node 47):** filter out the empty rows, based on crash\_date



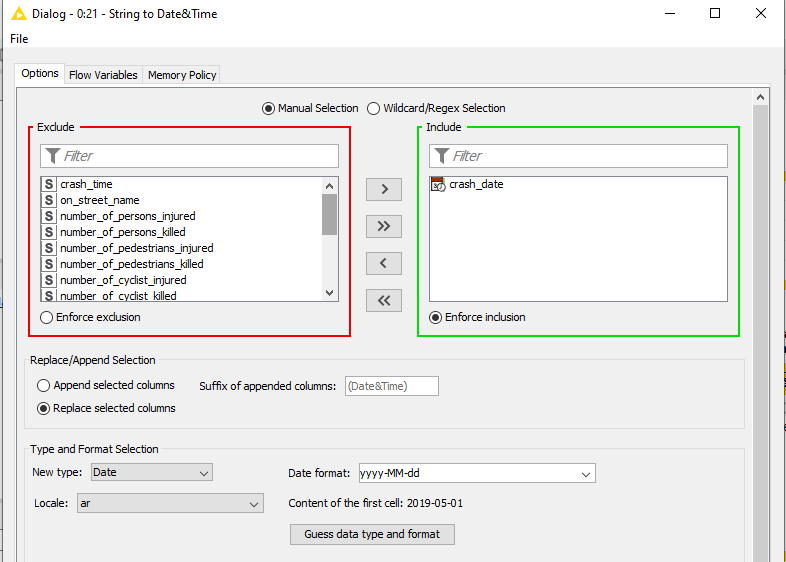
1. **String Replacer (Node 15):** Correct the crash\_date column, remove string “T00:00:00.000”



1. **String to Date/Time (Legacy) (Node 20):** Convert the **crash\_date** column to date

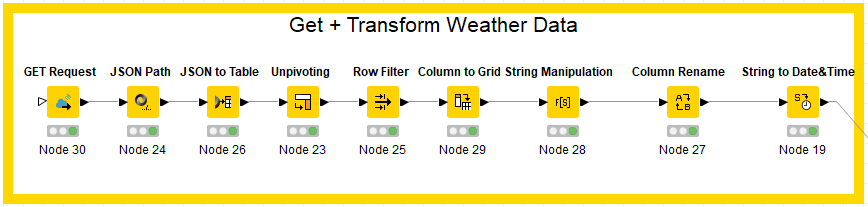


1. **String to Date/Time (Legacy) (Node 20):** Convert the **crash\_date** column to date



### Precipitation Data

#### KNIME Workflow



#### Data Description

* **date:** Occurrence date
* **precipitation:** Precipitation in milliliters

**PS: The data contains observation of precipitation levels for every day of 2019, additional attributes like wind and the station ID where the data was recorded. The LAGUARDIA AIRPORT, NY US (ID: USW00014732), was used as an exclusive source of the precipitation, due the availability of daily records of precipitation data.**

#### Workflow Description

1. **Get Request (Node 30):** calls data from the NOAA API for 2019, including precipitation data, attributes such as wind, station ID and date.

**Query Parameters:** h<ttps://www.ncdc.noaa.gov/cdo-w>eb/api/v2/data?datasetid=GHCND&startdate=2019-01-01&enddate=2020-01-01&units=metric&datatypeid=PRCP&limit=366&locationid=FIPS:36&stationid=GHCND:USW00014732

**Dataset ID:**

datasetid=GHCND

**Date Range:**

startdate=2019-01-01&enddate=2020-01-01

**Units:**

units=metric

**Data Type ID:**

datatypeid=PRCP

**Search Limit:**

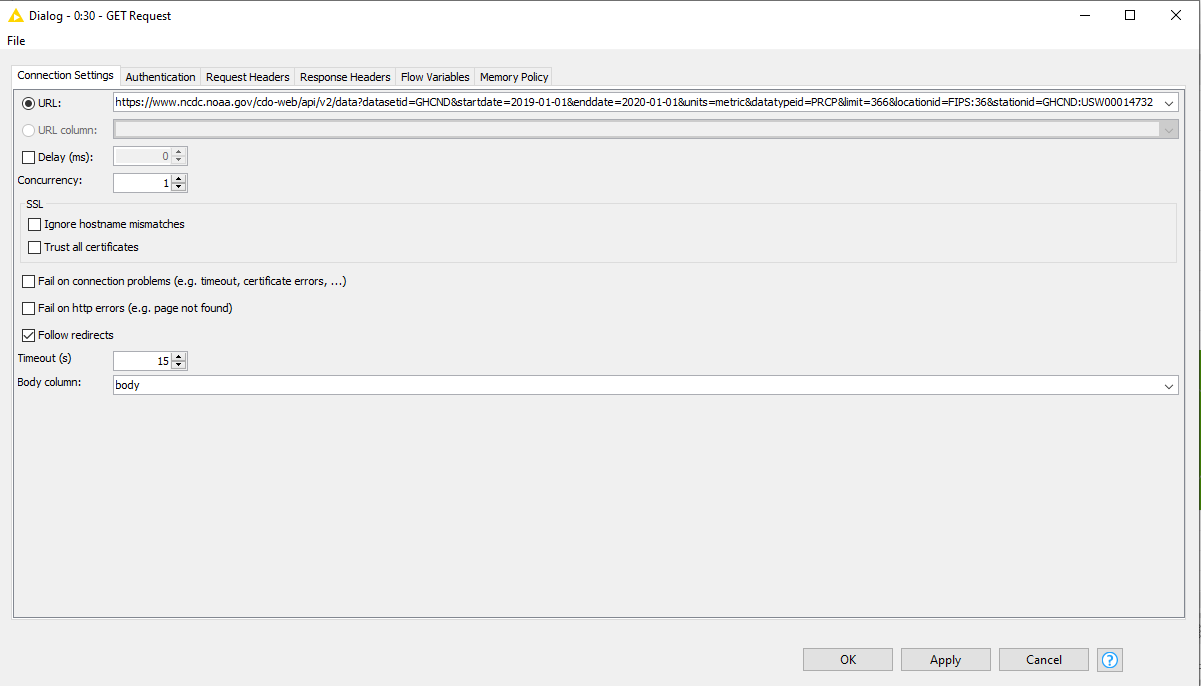
limit=366

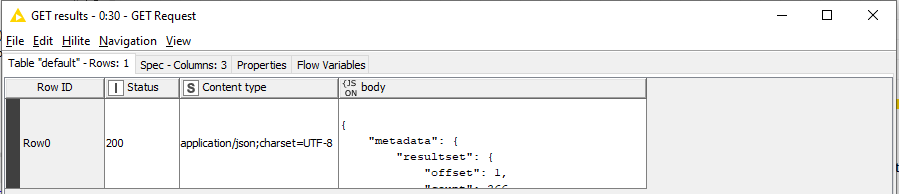
**Location and Station ID:**

locationid=FIPS:36&stationid=GHCND:USW00014732

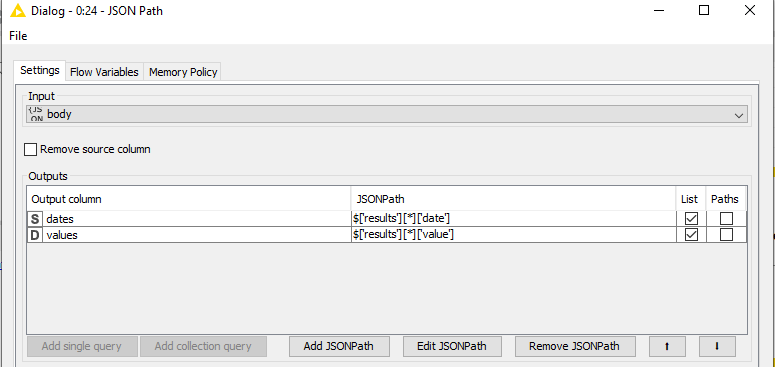
For more information, regarding the API, token generation and how to determine the parameters of the query please access:

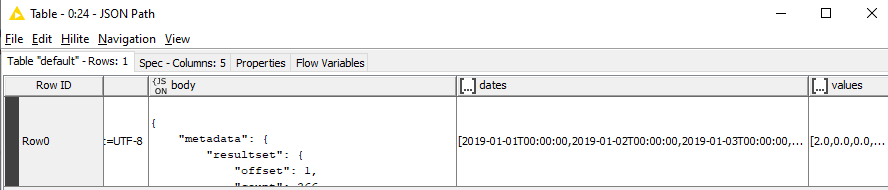
https://www.ncdc.noaa.gov/cdo-web/webservices/v2#datasets



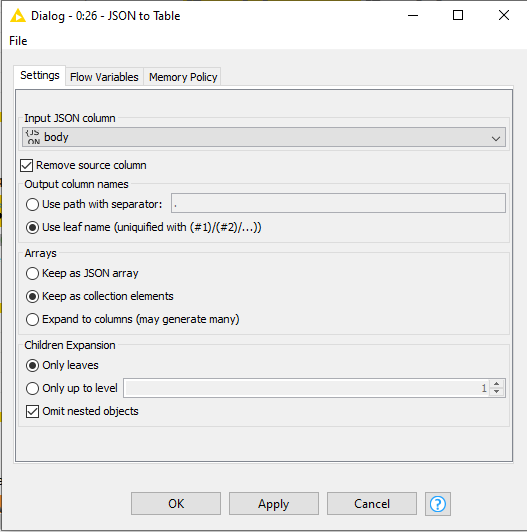


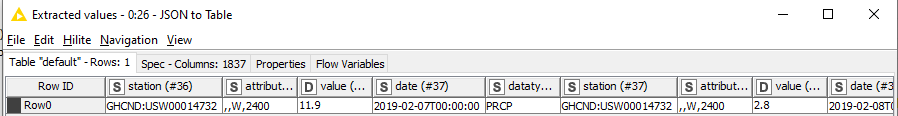
1. **JSON Path (Node 24):** queries the called data including the dates, precipitation, all the attributes and station ID



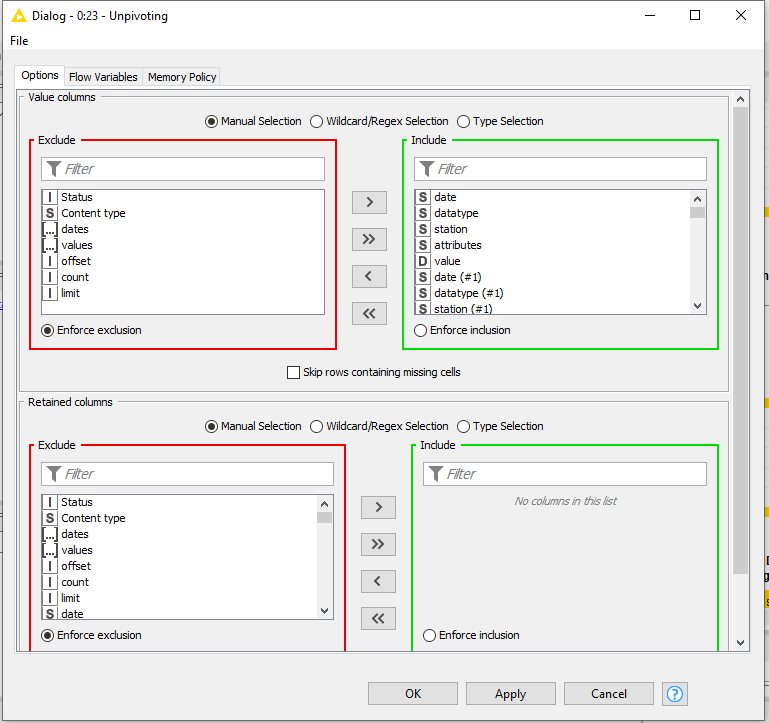


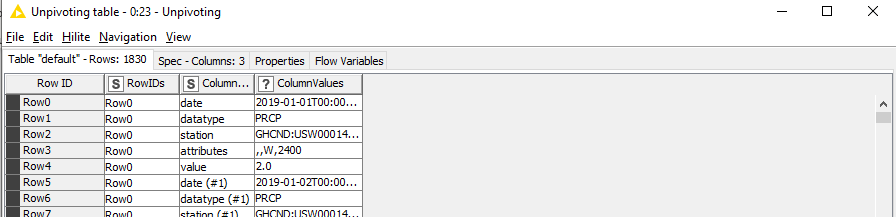
1. **JSON to Table (Node 26):** extracts the aforementioned data from JSON to a tabular format, yielding a table with one row



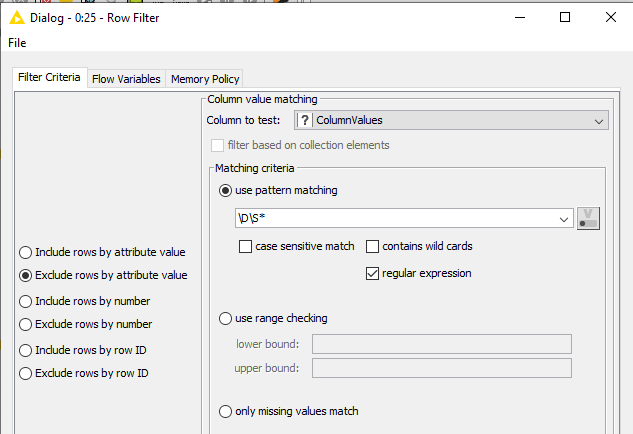


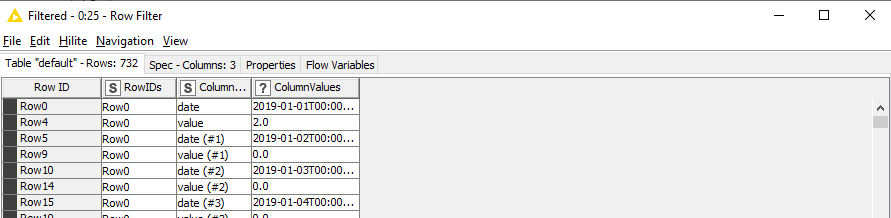
1. **Unpivoting (Node 23):** rotates the columns containing the called data from the input table to row and duplicates at the same time the remaining input columns by appending them to each corresponding output row. Results in a table that has multiple rows and groups the data column names and column values



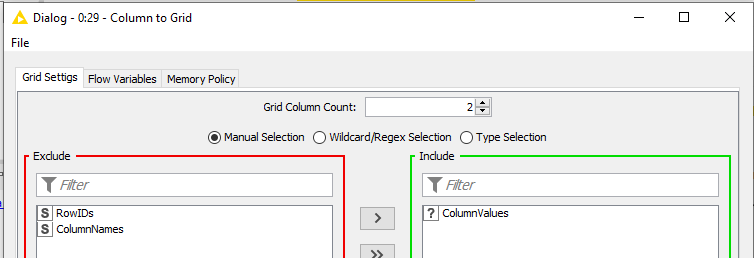


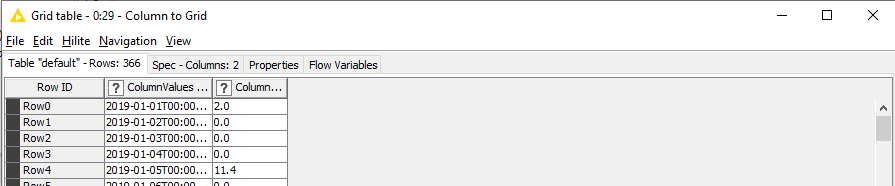
1. **Row Filter (Node 25):** filter out the redundant data, keeping only the precipitation data and date



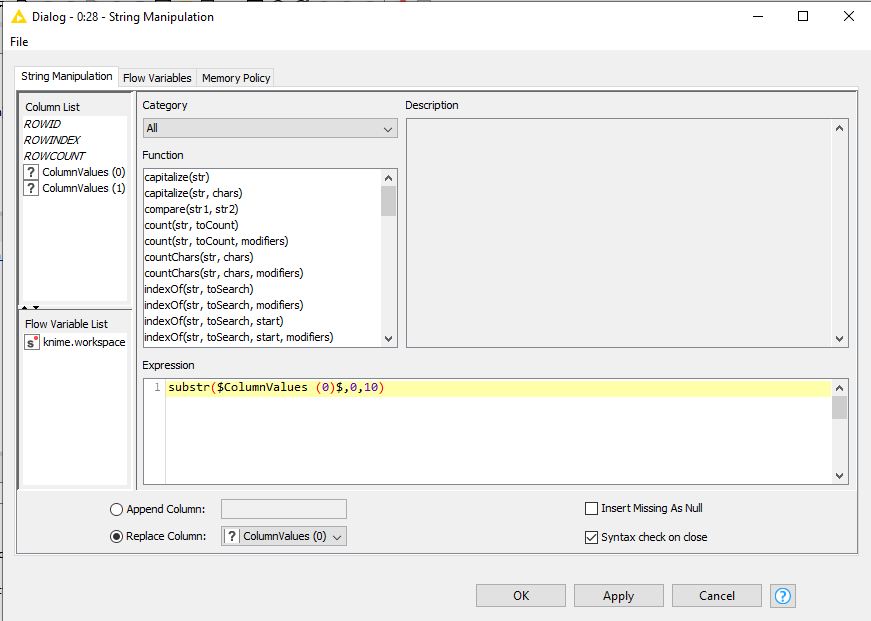


1. **Column Grid (Node 29):** breaks the column, containing the precipitation and date into two new columns, such that they align in a grid. Yields a table with separate column for date and precipitation data

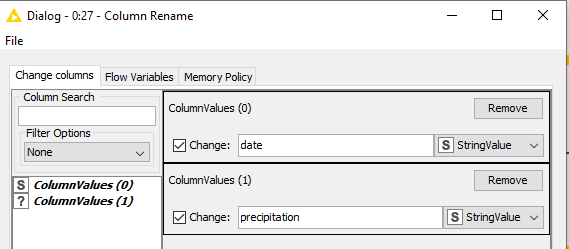




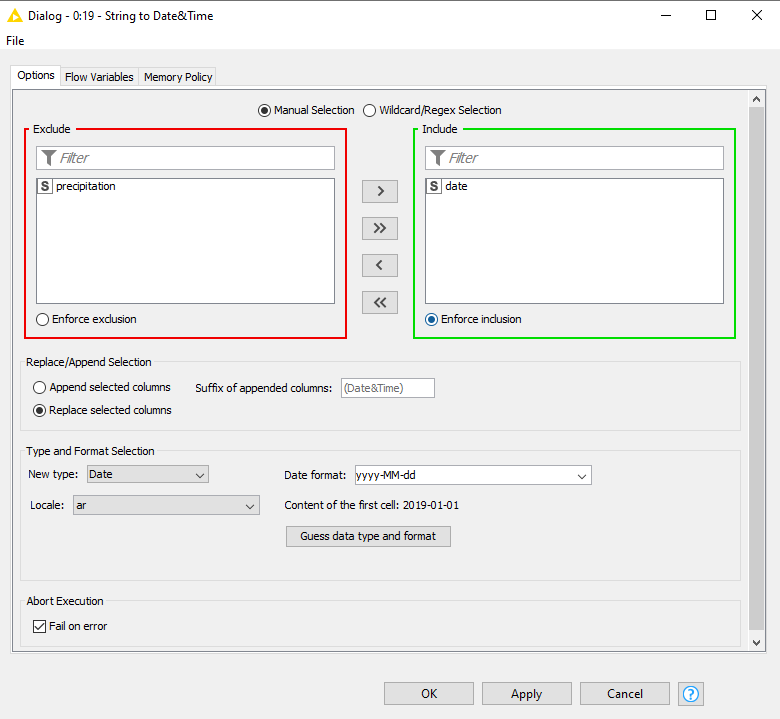
1. **String Manipulation (Node 28):** manipulates the strings in “date” column and removes the “T00:00:00” indicator of time

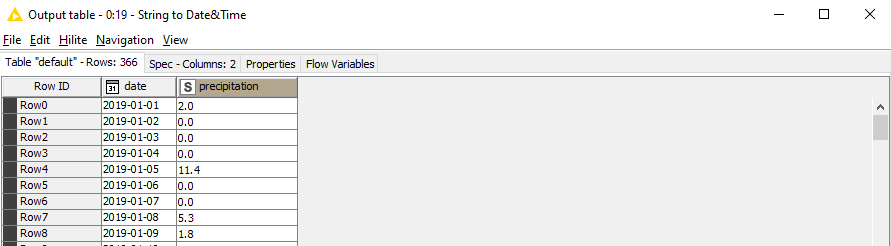


1. **Column Rename (Node 27):** renames the cleaned columns to “date” and “precipitation”



1. **String to Data/Time (Node 19):** Converts the data in the “date” column from string to date format



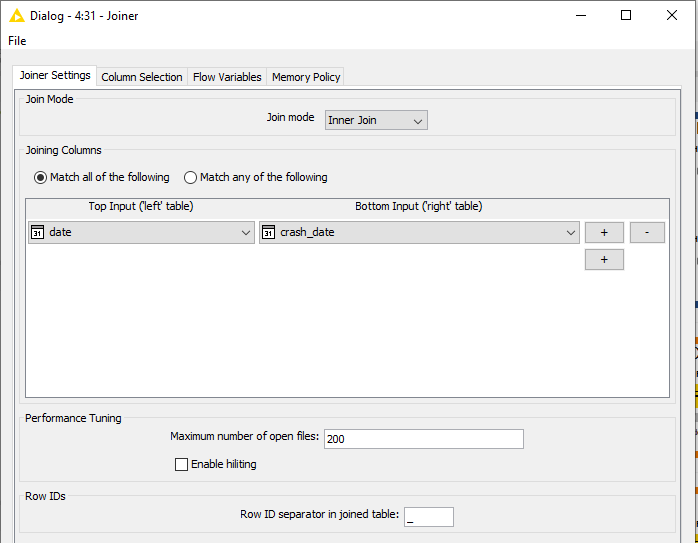


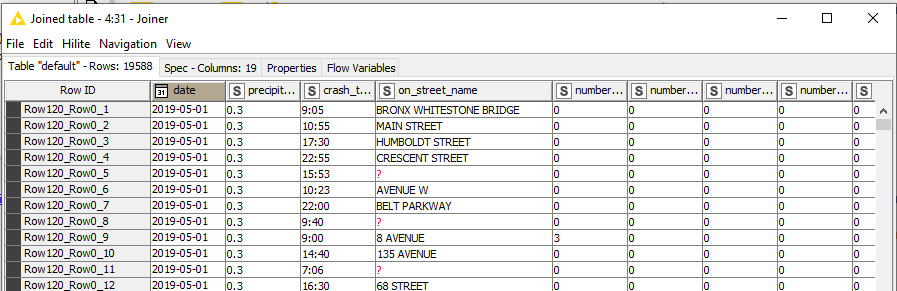
## Analytics

### Join + Fix Data Types

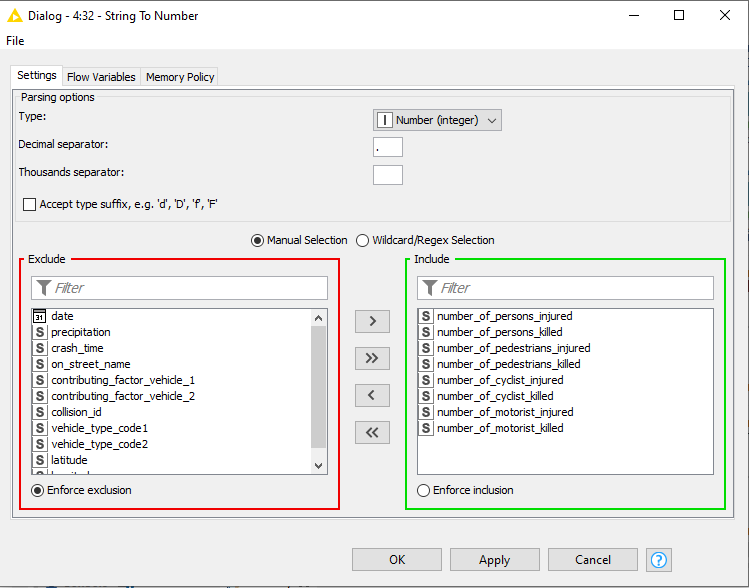
Once the datasets are cleaned and extracted it is time to join those tables into one for further analysis. The inner joiner will use the common date between the Motor Vehicle and Precipitation datasets.

1. **Joiner (Node 31):** Inner joins the dataset using the columns date (precipitation) and crash\_date (Motor Vehicle)

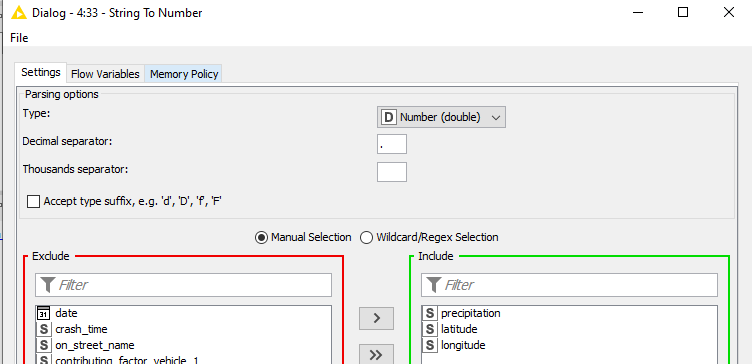




1. **String to Number (Node 32):** Convert the measuring variables into Integer.



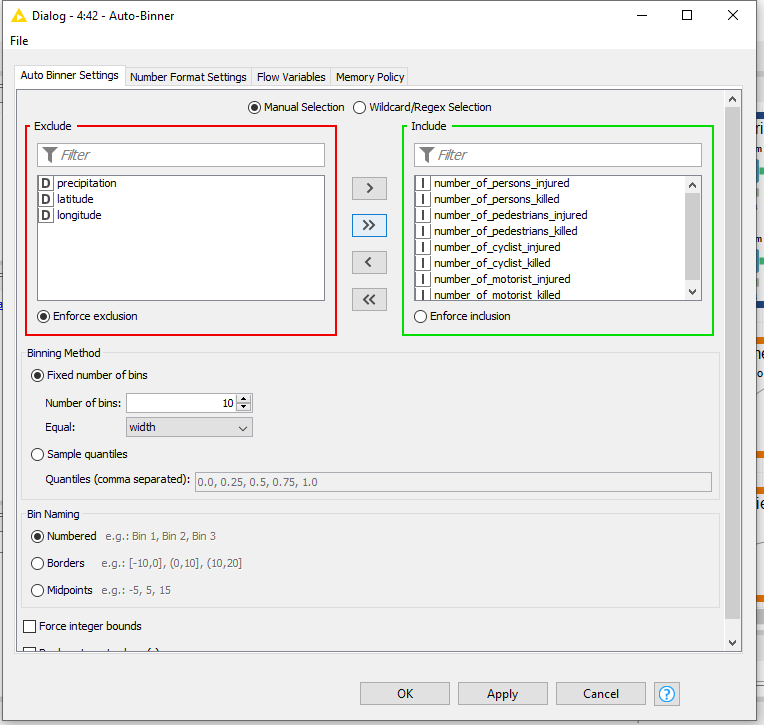
1. **String to Number (Node 33):** Convert the measuring variables into Double.



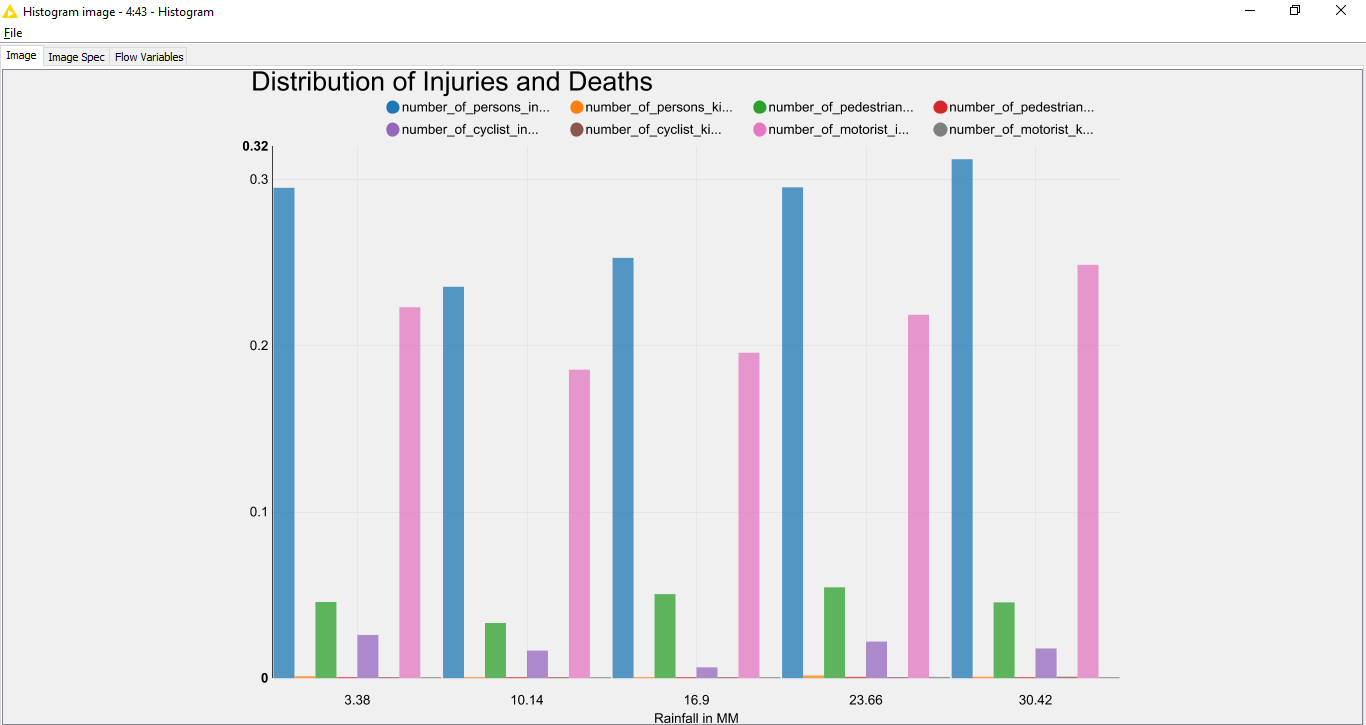
### Analysis: Histograms

After the datasets are joined and transformed, it is time to begin with some quick descriptive analysis. The distributions chosen were **Injury and Death** numbers

1. **Auto-Binner (Node 42 and 55):** sorts our variables into bins automatically, by creating new columns in the table with a bin number. This makes it easy to plot them on the histogram in the next step.



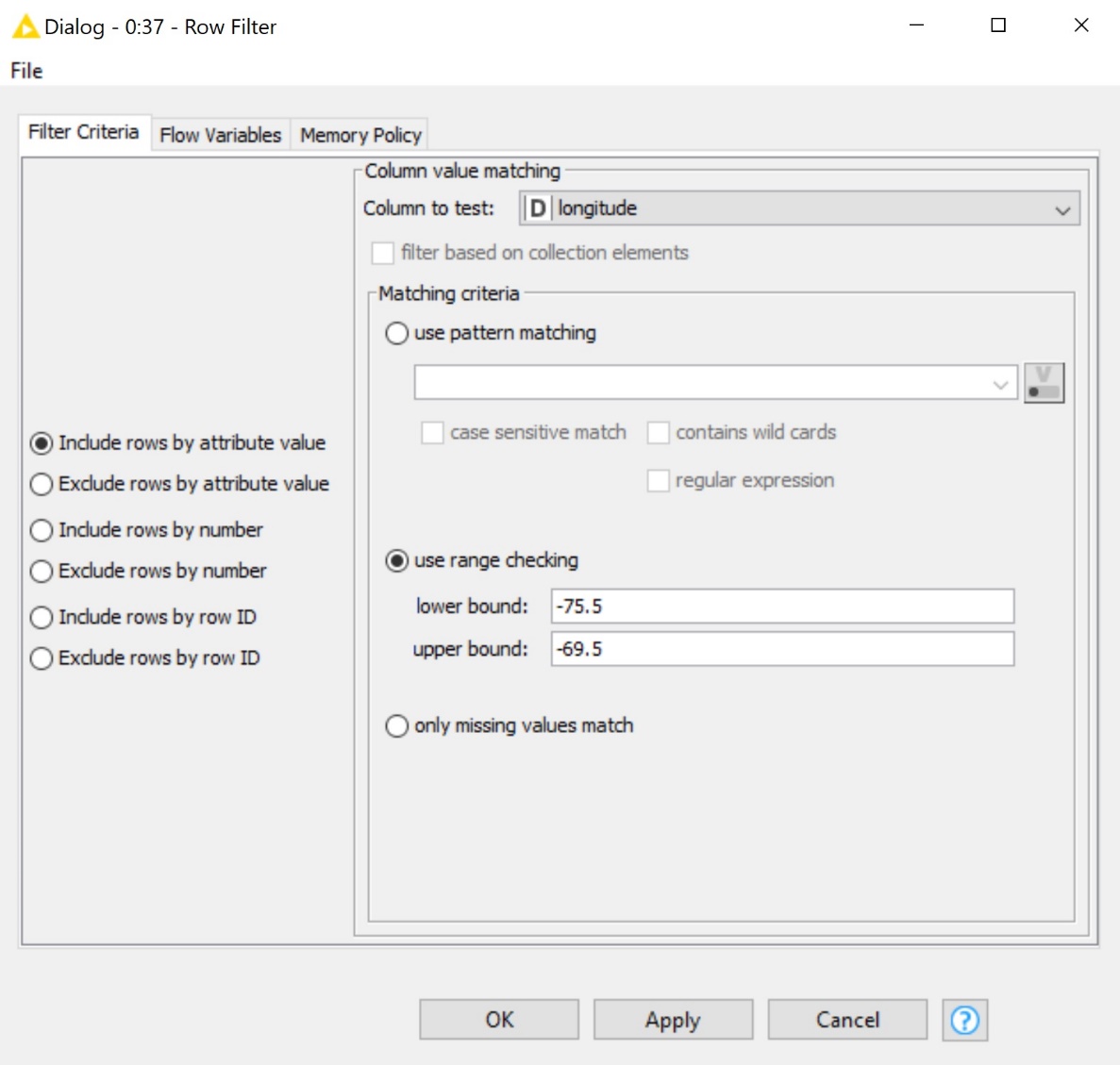
1. **Histrogram (Node 43 and 54):** show the distribution of different types of deaths and injuries conditioned on rainfall, and the distribution of all injuries conditioned on rainfall (only Node 43 is shown below)



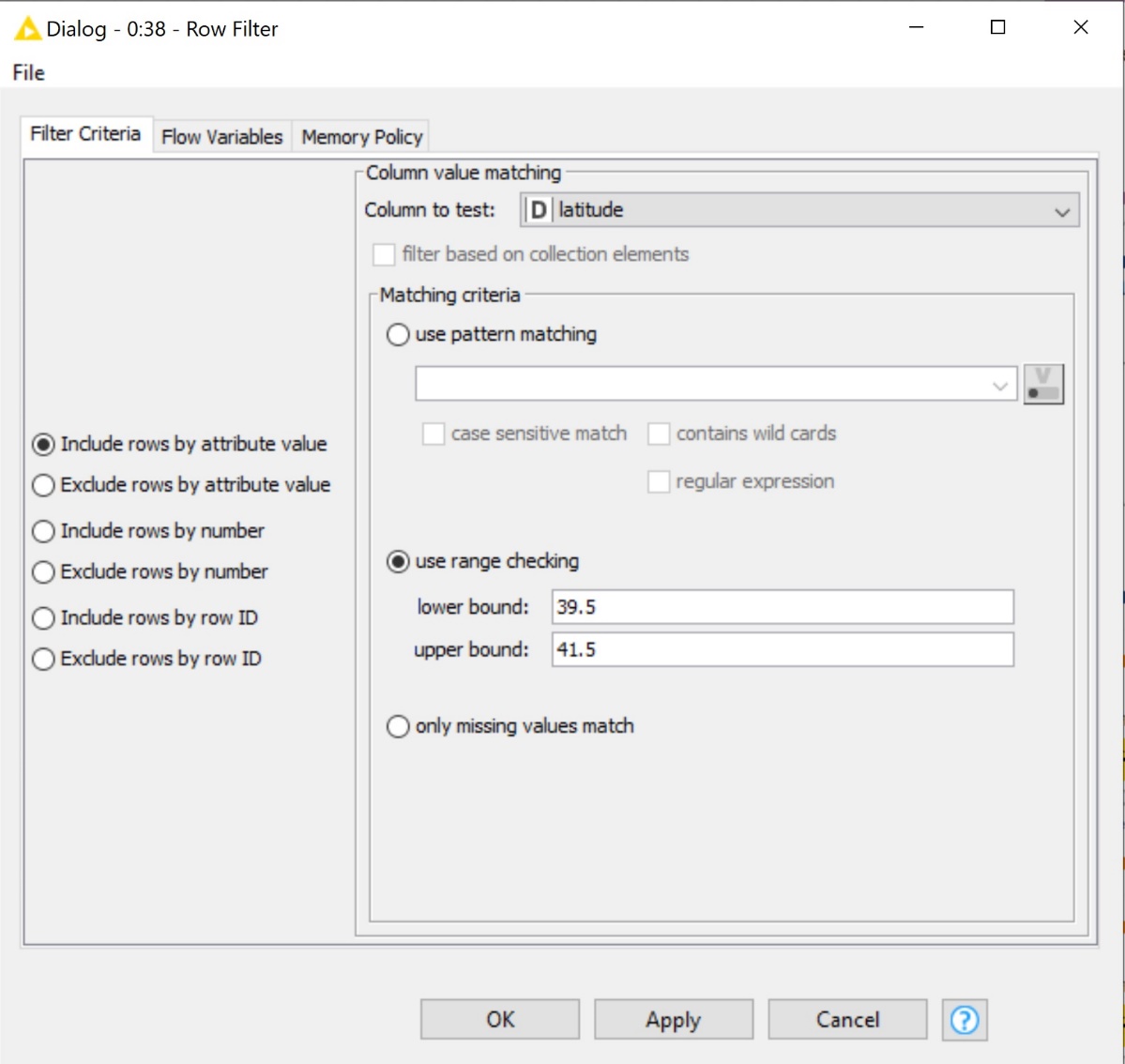
### Analysis: Maps

Another important analysis that could be also done is related to the accident’s location itself. Once the dataset includes the location of the crashes it´s possible to visualize through map where most of those accidents happened. The maps created were the **Lethal Crashes and Cyclist Injuries**.

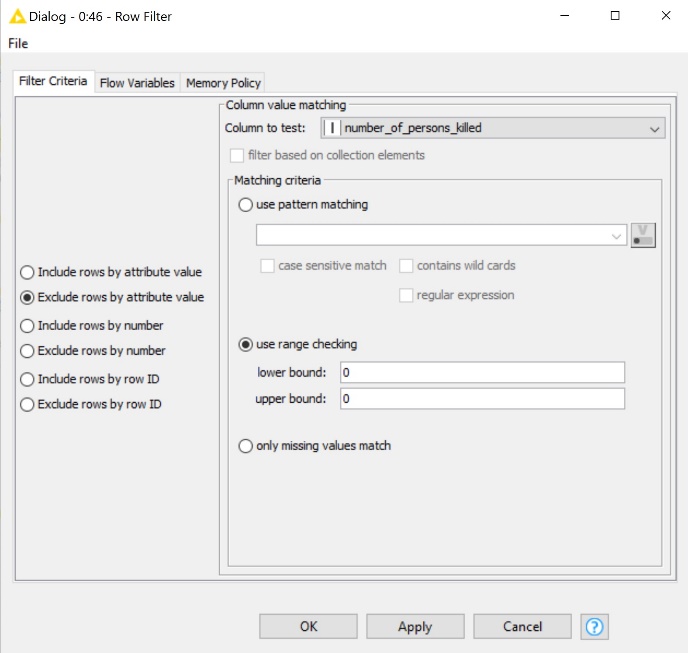
1. **Row Filter (Node 37 and 51):** filter the table so that it only includes **longitude** values which belong to New York City (this basically filters out typos in the longitude field).



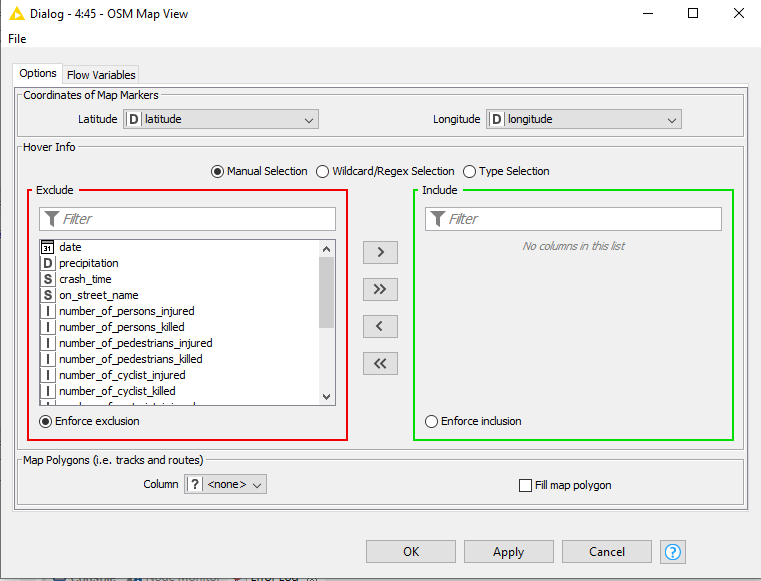
1. **Row Filter (Node 38 and 52):** filter the table so that it only includes **latitude** values which belong to New York City (this basically filters out typos in the longitude field).

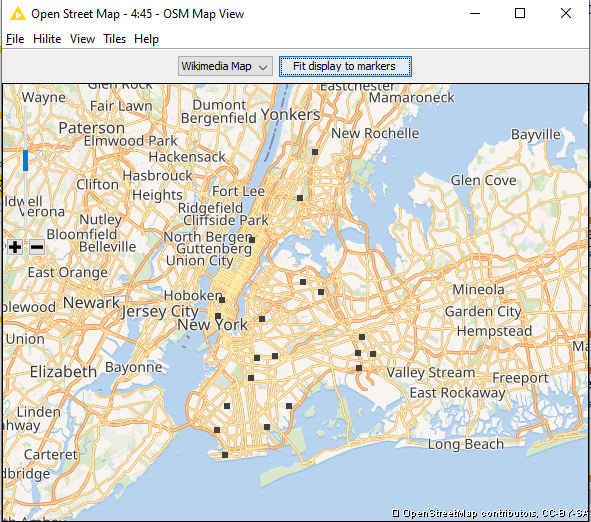


1. **Row Filter (Node 46 and 49):** filter the data to only include crashes which had **non-zero deaths** (Node 46) or **had a cyclist injured** (Node 49). (Node 46 showed bellow)



1. **OSM Map View (Node 45 and 50):** show every observation that met the filter criteria on a map using Open Street Maps. (Node 45 shown below)



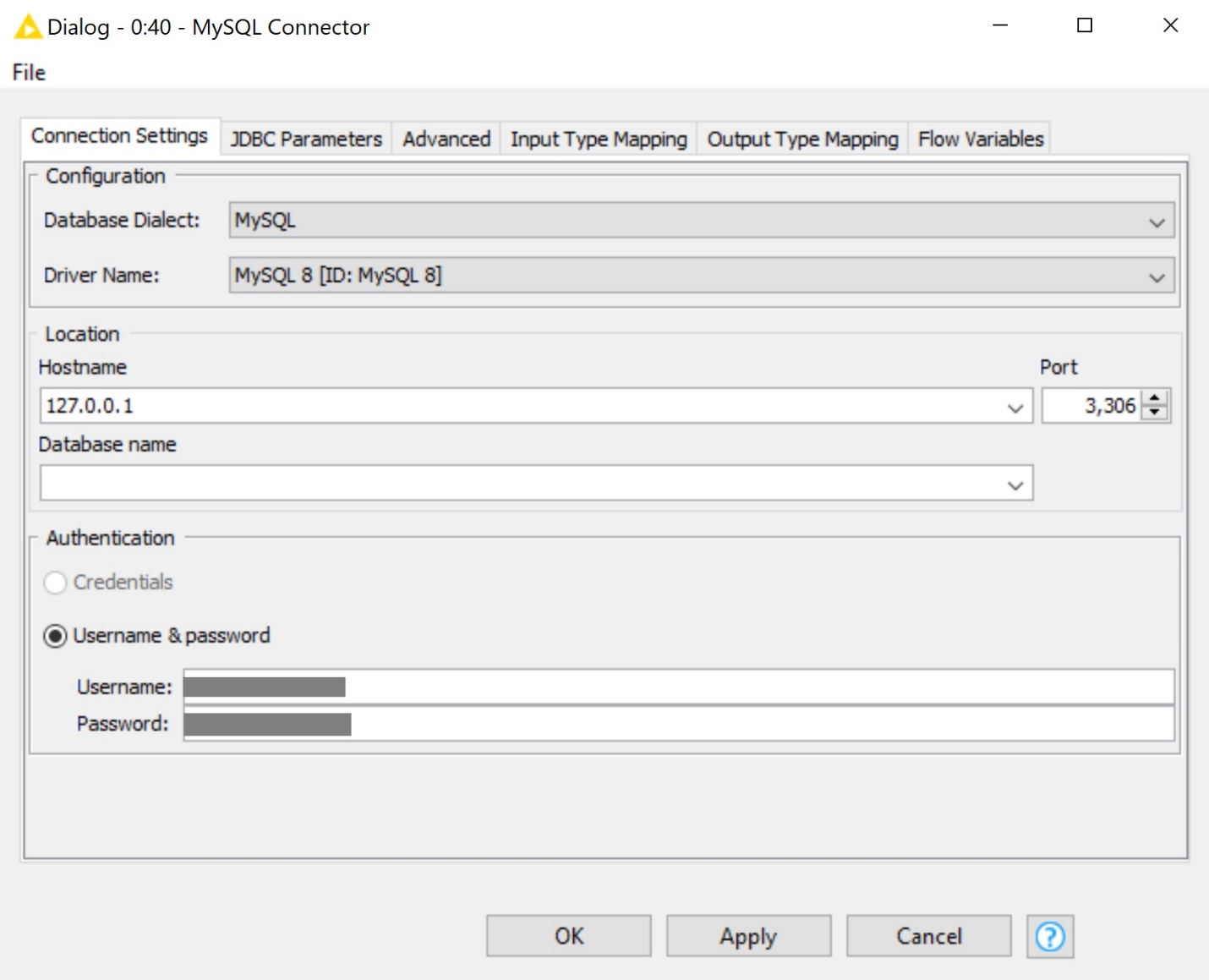


## Permanent Data Storage

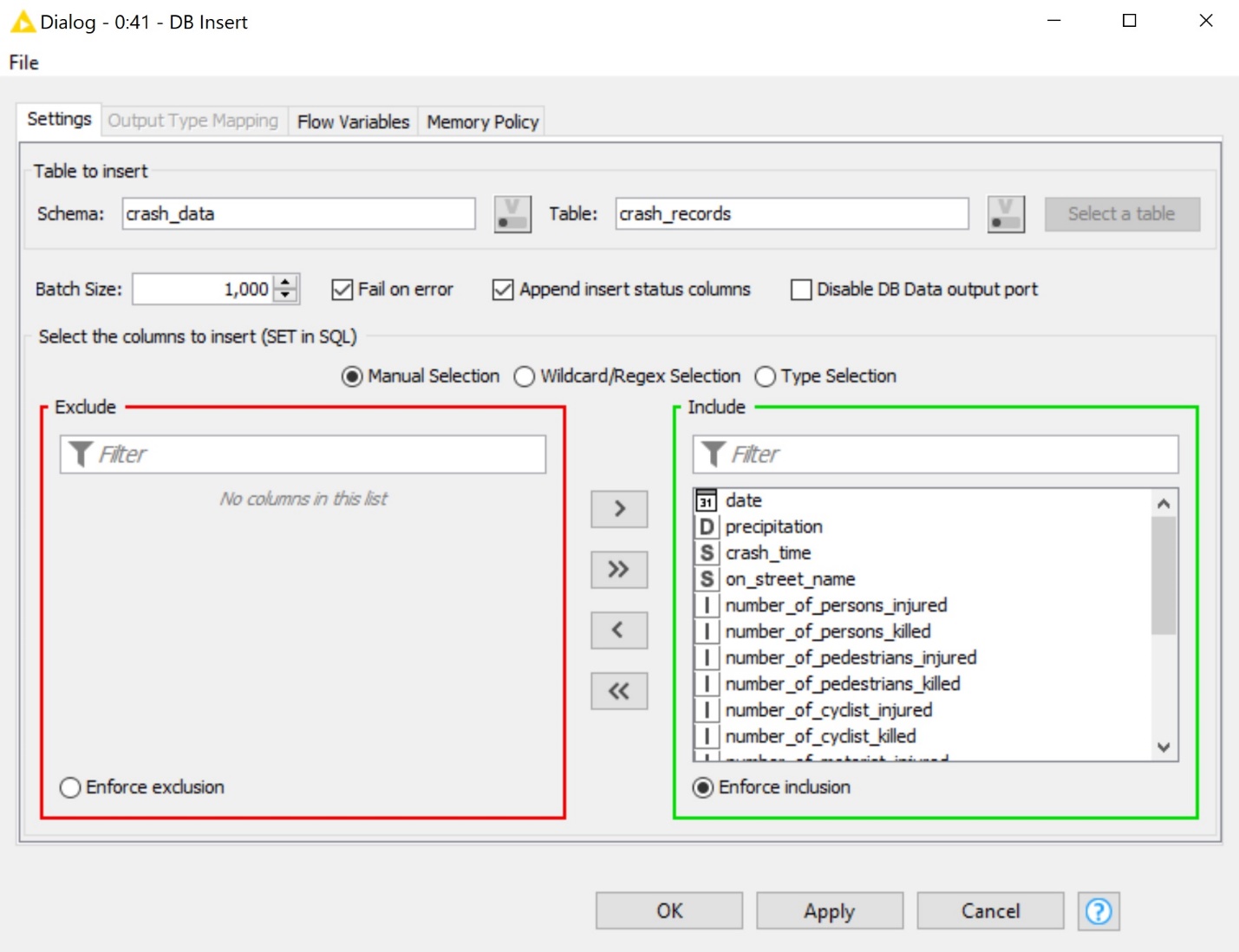
### MySQL Server

One of the ideas of the tool is also to maintain a local database with permanent records for more detailed analysis in the future. It was chosen to write out data to a local MySQL server, because this solution was simple and fast, and it required minimal configuration.

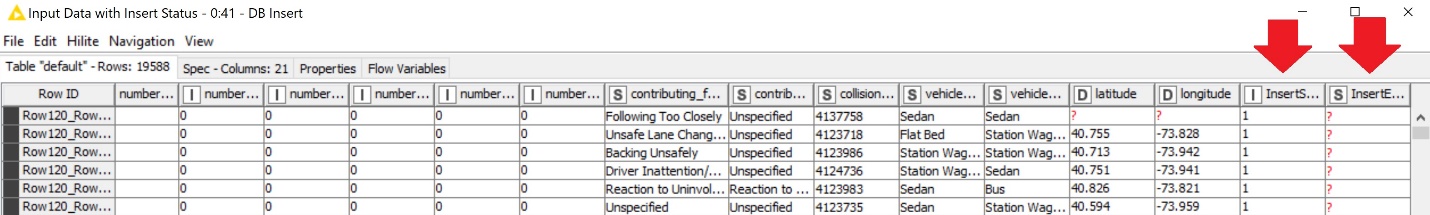
1. **My SQL Connector (Node 40):** connects to a local MySQL server and authenticates with given username and password. In our tool case, it´s connected to a local machine.



1. **DB Insert (Node 41):** inserts the supplied data into the specified schema and table on the MySQL server. **NOTE: column names must match exactly**



It´s possible confirm that the table was inserted by checking the DB Insert node and viewing the Input Data with Insert Status. If the value of InsertStatus is 1, it was successful. If there were an error for any row, it would show the error in the InsertError column.



## Conclusion

### Project / Tool Overview

The final perception about the project and the tool created were pretty much satisfactory. At the beginning it was defined monthly actualization of the tool and the datasets, however the data extraction part was even easier and faster than expected. Those periods can be even longer than one single month and, in that sense, it would be considering something bigger than the actual 20-30k monthly average observations.

Connecting and using API´s with KNIME showed itself absolutely user-friendly. Although the datasets differ a bit from each other, the process was pretty much the same with really small differences.

In the meantime, considering the aspect of the analysis and the data transformation, KNIME can be useful and bring some great insights about the data however, it´s still a bit rigid and has its own analysis limitations.

The transformations are done using a great number of nodes and the analysis do not go much further than the descriptive ones. It’s indeed possible to download extra packages for both, but the recommendation it’s to go for a more powerful analysis tool like R or Python

Finally, the conclusions taken from the dataset and his stability/reproducibility were significant enough. KNIME provided a user-friendly structure that could deal well with the connections and a big amount of data that allows it to be useful for a bunch of cases, like this one. It may have it’s limitations regarding the analysis possibilities, however it produces a reproduceable and stable ETL process and datasets.